Automated Conversion from Spreadsheets to Database

by

Ankur Bajoria

A Project Report Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in
Computer Science

Supervised by

Dr. Xumin Liu

Department of Computer Science

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

May 2014

The project “Automated Conversion from Spreadsheets to Database” by Ankur Bajoria has been examined and approved by the following Examination Committee:

Dr. Xumin Liu
Professor
Project Committee Chair

Dr. Rajendra Raj
Professor

Dr. Trudy Howles
Professor

Abstract

Automated Conversion from Spreadsheets to Database

Ankur Bajoria

Supervising Professor: Dr. Xumin Liu

Spreadsheets have been around for years and are considered the standard documentation format when dealing with data in a tabular format. A spreadsheet is designed to work much like a database, it has a cell like structure with a cell being a member of a particular row and corresponding column. When compared to a database a spreadsheet lacks many features that makes it less appealing for use in data storage and processing. A spreadsheet suffers from data redundancy, where multiple copies of a similar data instance may exist in the same spreadsheet document. A spreadsheet does not have the capacity to provide multiuser access like a database and it also has limited storage capabilities. By converting a spreadsheet document into a database table a user can leverage all the advantages of a database on their existing data. This project includes a discussion on the differences and similarities between spreadsheets and databases, a framework to convert from a spreadsheet to a database table and back. The project then implements the framework, thus defining a structure for excel spreadsheets and giving it all the missing features that a database has. The project also takes a look at a popular algorithm to find functional dependencies between the columns of the created database table and utilizes an implementation of the algorithm to find functional and transitive dependencies on the columns in the database.

Contents

Abstract . . . . . **iii**

1 Introduction **1**
1.1 Background 2
1.2 Related Work 7
1.3 Problem and Hypothesis 8
1.3.1 Problem Statement 8
1.3.2 Hypothesis 8
1.4 Solution 9
1.4.1 Hypothesis Validation 9
1.5 Roadmap 10

2 Design **11**
2.1 Primary Key and Data Type Prediction 11
2.1.1 Importing an Excel Sheet 11
2.1.2 Exporting a Database Table 14
2.2 Functional and Transitive Dependencies 14
2.2.1 TANE Algorithm 14
2.3 Application Flow 16
2.4 User Input 17

3 Implementation **19**
3.1 Languages and Tools 19
3.2 Class Overview 19
3.2.1 Excel Component Overview 19
3.2.2 TANE Component Overview 21
3.3 User Input 22

4 Analysis **26**
4.1 Testing Environment 26
4.2 Datasets 26
4.3 Import 28
4.4 Export 29
4.5 Hypothesis Evaluation 31

5 Conclusions **34**
5.1 Current Status 34
5.2 Future Work 34
5.2.1 Support For Newer Versions of Excel 34
5.2.2 Faster Data Prediction Algorithm 35
5.2.3 Support For More Databases and Data Types 35
5.2.4 Additional User Experience Changes 35
5.3 Conclusions 35

Bibliography **37**

A UML Diagrams **38**

B Code Listing **41**

C User Manual **42**
C.1 Installation 42
C.2 User Interaction 42
C.3 Error Handling 45

List of Tables

2.1 Example database table 15

List of Figures

2.1 Process Flow for GUI application. 18

3.1 High level class diagram for the Excel Component. 20
3.2 High level class diagram for the TANE Component. 21
3.3 GUI application on first start. 23
3.4 GUI application when importing a spreadsheet to the database. 24
3.5 GUI application displaying output for Functional and Transitive dependencies. 24
3.6 GUI application when exporting a database table to an excel spreadsheet. . 25

4.1 Time Measurements for the Import operation. 30
4.2 Time Measurements for the Export operation. 30
4.3 Database Table on the H2 Database Console. 33
4.4 Functional and Transitive Dependencies for the Database Table. 33

A.1 Excel Component Class Diagram. 39
A.2 TANE Component Class Diagram. 40

C.1 GUI application on first start. 43
C.2 GUI application when importing a spreadsheet to the database. 44
C.3 GUI application displaying output for Functional and Transitive dependencies. 44
C.4 GUI application when exporting a database table to an excel spreadsheet. . 45
C.5 Import Error. 46
C.6 Export Error. 46
C.7 Multiple Primary Keys Chosen Error. 47
C.8 Invalid Primary Key Error. 47

Chapter 1

Introduction

A spreadsheet document is meant to handle small scale data in comparison to a database. Spreadsheets enable an end user with no application programming experience to store and manage a dataset of limited size. Spreadsheets are most widely used in the financial domain for accounting purposes owing to their ease of use and support for reporting tools in the form of graphs and charts. The data is normally represented in the standard numeric or text data type for each column. A spreadsheet provides additional support for a user to define the data in a cell using a formula. A formula is a relationship between two or more columns with an aggregation operation separating each column.

A database has the same structure as a spreadsheet in terms of rows and columns but is designed to provide storage for large scale data. It also provides better data integrity and consistency. Databases reduce the amount of data redundancy using normalization. This protects data from various factors like insert, update and delete anomalies. They allow various users to access the data in the tables while keeping the data characteristics intact. A database is designed to process high level queries on the stored tabular data. This is done using high level query languages like SQL. A user needs to have certain amount of programming background before using SQL to query a database.

Spreadsheet documents lack a definite structure. For example consider a spreadsheet for storing employee information, the spreadsheet has columns that specify the employee ID, name and salary. The ID and salary are numeric fields as you would expect while the name

stores data in the form of characters. A spreadsheet document however does not explicitly force these rules, the name field could be populated with a numeric value and the ID or salary fields could hold characters. When trying to convert a spreadsheet into a database this rule is of critical importance since a database strictly enforces data type rules for its columns. Databases also support the use of primary keys. A primary key is a column in the database that stores unique values for each row and can be used to distinguish between rows. Spreadsheets on the other hand do not have a concept of primary keys. In the case of the above example the ID field would ideally be used as the primary key. However a spreadsheet allows redundant data to be present in more than one row, this would mean that more than one employee would have the same ID which is not possible in the real world. This proposal focuses on extracting data from a spreadsheet and converting it into a database table by determining the data types of the columns and in the process defining a structure to the spreadsheet data.

1.1 Background

There are various applications that provide creation and manipulation of spreadsheet documents namely Microsoft Office Excel, OpenOffice Calc, LibreOffice Calc of which Microsoft Excel is the most widely used. Microsoft Excel was designed as early as 1985. It was designed to facilitate record keeping and updation for the financial industry. Microsoft Excel follows the same standard structure in terms of how data is stored in excel files. The data is defined in a grid structure. Where each cell of the grid is defined by its corresponding unique row and column number. Each grid represents what is called a Worksheet in Excel. There can be many such worksheets included in a single Workbook. A workbook represent a folder like wrapper to hold the different worksheets that might be related to each other. Each column in the spreadsheet can hold data of a different data type. The most common data types defined in Microsoft Excel are number, text, date, currency. Number is used to store numerical values while text stores string values like names, addresses etc. The date data type is used to store the date which by default is M/d/yyyy. The currency

data type stores data in the form of numbers with the optional currency symbol preceding the value. The symbol however has to be manually specified.

Databases have been broadly classified into two major types namely SQL and NoSQL (Not only SQL). The major differences between the two types of databases is the fact that an SQL database follows a proper schema or structure for the tables defined, whereas following the NoSQL approach would mean that there would not be any defined structure for the database tables. Another distinguishing factor between the two types of database systems is the characteristics that they follow. SQL databases follow the ACID properties which are described as follows. ACID stands for Atomicity, Consistency, Isolation and Durability.

- Automicity states that either a transaction in the database must commit properly or in case of failure no changes made by the transaction should persist.
- Consistency states that changes made by every transaction in the database must keep the database in a valid state.
- Isolation means that when multiple transactions are executed in the database the end result should be similar if the transactions were executed in serial order.
- Durability simply states that the results of committed transactions must persist in the database in the event of failures.

A NoSQL database however does not follow the same properties as an SQL database. A NoSQL database typically follows the BASE properties. The BASE properties are defined as Basically Available, Soft State and eventual Consistency.

- Basically Available refers to the availability of data on the event of failure. For

example the failure of a single node should not effect the availability of data in the database.

- Soft State refers to the state of the system at a given time, which may change based on the propagation of updates.
- Eventual Consistency means that the database will not guarantee that the database will be in an updates states after every transaction, instead the updates will propagate through the database over time.

In order to define a structure for the spreadsheet the project will make use of a SQL database that follows the ACID properties such as the H2 database. The H2 Database is an open source project under the Eclipse Public License. The database it written entirely in JAVA and is designed to be very fast. H2 database has support for a wide range of data types, some of the most prominent ones are int, double, real, char, varchar, blob etc. H2 database is relatively new but is widely accepted owing to the inherent JAVA integration and faster speeds when compared to other relational database systems. It also has a lot of built in security features.

The data needs to be pre-processed before it is fit for conversion into a database table. The pre-processing includes cleaning of data and its validation. Cleaning of data here refers to removal of bad data, for example a column ID that stores values about a employee's ID cannot contain special characters in it. Such data has to be removed from the spreadsheet. The cleaning phase will check for such inconsistent values in the spreadsheet and warn the user about their presence. The pre-processing phase is also responsible for dealing with the missing values in the file. These values can either be imputed and replaced with appropriate data or be considered as blank when converting into a table. The missing values are typically replaced with the mean or mode for numeric values. For nominal data the value that is present maximum number of times in the column can be used.

Functional dependencies are relationships between one or more columns of a database table. By definition a functional dependency between two columns or attributes is present when one column can uniquely determine the values of the other. This is typically represented in the form of X $\rightarrow$ Y which means that every value in X is associated with only one value in Y. Here X is also called the determinanat and Y is called the dependant. For example, consider a database table with two attributes Social Security Number (SSN) and Name. In this case we can define a functional dependency SSN $\rightarrow$ Name. Since every person will have a unique SSN and given the SSN you can look up the Name from the database table, it can be used as the determinant while Name will be the dependent. Such a functional dependency will be termed as a trivial functional dependency. There can be multiple attributes that make up the determinant in a dependency. These are called as multivalued dependencies. They are of the form X,Y $\rightarrow$ Z. In such a multivalued dependency if the functional dependency still holds when an attribute is removed from the determinants then it is not a full functional dependency. When there are two functional dependencies of the form A $\rightarrow$ B and B $\rightarrow$ C, then we can imply that A $\rightarrow$ C, this is known as a transitive dependency.

There are different types of anomalies found in a database table. It is important to find these functional dependencies in a database in order to remove anomalies from it. The different types of anomalies found in databases are,

- Insertion Anomaly: An insertion anomaly occurs when there is a column that has values depending on another attribute. For example if there is a database table that holds information about cars, the attribute model name depends on the attribute company, so if there were to be a new insertion in the model name column without a corresponding entry in the company column then that would be an insertion anomaly.

- Updation Anomaly: An updation anomaly occurs when there are several entries of the same data present in different tables and if there is an update on one of the entries

then there should a subsequent updates in the other tables.

- Deletion Anomaly: A deletion anomaly occurs when there is data left behind from an improper deletion for example if there is only one entry in the table for a particular car company and its corresponding model and if the model is deleted then the car company entry has to be deleted as well.

The process of avoiding these anomalies is called normalization. When applying normalization to a database table, the table is broken down into smaller more managable tables. The smaller tables are all connected by the relationships defined between them. There are various levels of normalization, the most commonly ones used are 1NF (First Normal Form), 2NF (Second Normal Form), 3NF (Third Normal Form) and BCNF (Boyce-Codd Normal Form).

The project makes use of an efficient algorithm called TANE [2] to determine the minimal functional dependencies from the created database table. Based on the output of the algorithm the framework determines the transitive dependencies for the table. The concept behind the TANE algorithm is to divide the tuples or individuals rows of the table into partitions, hence creating smaller sets of rows. It does this partitioning based on the attribute values of the rows. This makes the algorithm efficient in the sense that it can test the presence of the functional dependencies for larger tables much faster. The TANE algorithm also identifies the presence of approximate functional dependencies in the table. To discover the functional dependencies the algorithm looks at a set of tuples that have some set of attributes in common.

1.2 Related Work

There has been considerable research in the field of spreadsheets and databases inter connectivity with many papers being published about it. Most research though has not been targeted at the conversion of spreadsheet to databases or vice versa but instead to add frameworks or plugins into spreadsheets and database systems to make them behave like the other.

The authors of paper [5] suggest the use of extensions to the SQL language itself in order to process spreadsheet like queries in an RDBMS environment. These SQL extensions would in turn provide array based calculations to help create models like spreadsheets. The authors have implemented these SQL extension to be used primarily with ROLAP databases. ROLAP which stands for Relational Online Analytical Processing is used to analyse data using multidimensional model. The SQL extensions help in ROLAP will help to create such multidimensional data models in the same as they would be possible in a spreadsheet. The authors of paper [4] suggest using a spreadsheet as a relational database table by constructing the operators that are used in relational algebra with the help of functions present in spreadsheets. The authors use standard worksheets in spreadsheet software like Microsoft Excel to store data and design formulas that act as queries. Another paper that takes a look at a different approach towards the problem. The authors of paper [3] develop a spreadsheet formula based language that defines structure to the existing spreadsheet data to convert it into the target format. The target here could be a database, a web service or any third party application that works on spreadsheet data. All these approaches however suffer from the fact that a spreadsheet does not provide the scalabilty or security that a database system does. The authors of paper [1] devise a solution to this problem by implementing a framework that can be used to convert data to and from a spreadsheet and a database. The framework works by defining the functional dependencies between the data columns of a spreadsheet, thus removing redundant and irrelevant information in the process. The framework then examines the functional dependencies to construct the

appropriate relational database schema.

1.3 Problem and Hypothesis

1.3.1 Problem Statement

When converting a spreadsheet to a database table it is very important to define a structure for the spreadsheet. A spreadsheet inherently does not define any structure to its existing data. This makes it difficult to define a schema when converting to a database table. The important data required here is the data type of the columns from the spreadsheet that is being converted. Typically this information can be obtained using various methods in many programming languages that would return the data type of the spreadsheet column. However spreadsheets have a very limited classification scheme for data types. The data types included in Microsoft Excel 2010 for example are numbers and strings. The numbers support values of type double and int while the strings can hold char values. A relational database system on the other hand has support for a large number of data types. The H2 database for example classifies integer values into a more specific buckets like smallint, mediumint, bigint. Apart from integer the H2 database also has support for double, real, decimal and binary numeric types. The problem of mapping of data types has been tackled before but the process is usually very generic and uses a set of predefined rules to enable the conversion. This project deals with the problem of mapping spreadsheet data types to database data types before they can be converted.

1.3.2 Hypothesis

The inherent problem of not having a structure in a spreadsheet makes it difficult when converting as spreadsheet to a database table. Without defining the data types the schema for the database cannot be defined and hence the spreadsheet cannot adhere to a specific structure. It is possible to define a pseudo structure for the spreadsheets if the data types

for the columns can be determined dynamically. This involves using a framework that will help determine the data types based on little or no previous knowledge of the spreadsheet columns. For the purpose of this project the framework will be used to determine the data types of the columns in the spreadsheet and create corresponding database columns.

The hypothesis of this project is that the use of a framework that uses range based comparison on the data entity in each column will enable the conversion of a spreadsheet to a database by providing correct mapping of the current spreadsheet column data types to existing data base types and in the process allowing the spreadsheet to be transferred to the form of a database schema. The framework will also take into account the formula columns defined in the spreadsheet. The data type for the formula columns will also be predicted using the range based comparison framework. The conversion will help define a structure to the otherwise unstructured data and also acquire database features like scalability, security and multi user access. The hypothesis also includes an implementation of the TANE algorithm to determine the functional dependencies between the columns in the created database. These functional dependencies are then used to determine the transitive dependencies between the database columns. The dependencies can then be used to normalize the database table. This will enable the database to avoid the various anomalies that leave the database in an inconsistent state.

1.4 Solution

1.4.1 Hypothesis Validation

The hypothesis will be validated based on the framework's accuracy. The accuracy will test the correctness of predictions for the data types of the spreadsheet data columns. The predicted data types can be compared to data types produced from an existing dataset with descriptions for the data types. The project can use these existing descriptions to compare

the predictions obtained through both the developed algorithm for the same dataset. In order to further validate the hypothesis the solution can be tested using multiple spreadsheets documents which have contrasting values in them. The validation will involve comparing the result under such different scenarios. The accuracy of the algorithm can not be quantified since it can not be measured as a metric as the output of the algorithm is just the data types of the columns. The final predictions will be compared to the output of the existing descriptions on the dataset. The column names of the spreadsheet will define the actual data types for the prediction of values. The project will also determine the time complexity of the data prediction component for the framework. This can be done by measuring the time taken for the import and export operations and examining the trendlines for the the same.

The accuracy of the functional dependencies found through the TANE algorithm will also be examined to determine whether the dependencies actually exist. If there is a functional dependency between two columns of the database table then they cannot contain multiple values that map from the determinant to the dependent. The functional dependencies can be verified by observing the semantics for the columns in the data. By understanding the meaning behind the data in the column the validity of the functional dependencies can be verified.

1.5 Roadmap

Section 2 describes the design of the framework. Section 3 describes the implementation for the framework including the implementation of the TANE algorithm used. Section 4 analyzes the accuracy of the framework and TANE algorithm implementation using different datasets. Section 5 discusses the conclusions and future work.

Chapter 2

Design

2.1 Primary Key and Data Type Prediction

2.1.1 Importing an Excel Sheet

The design for the project includes a standalone JAVA based application used to convert a given spreadsheet into a database table. The application is designed to take as input the file locations for the spreadsheet along with the database details to open the database connection. The details include the database name, the username and password to make a database connection. When importing the excel spreadsheet the application uses a path to the Excel file that a user can input on the file to import field.

Before the user can import the spreadsheet into the database, the spreadsheet has to be examined for missing and invalid data. The missing values pose a data consistency problem. The problem however with imputing values is that the there is data loss associated with it, the original data will loose meaning since imputation of values is done using either the mode and mean for numeric values or by using the values with the most occurrences in the case of nominal or string values. The data loss depends on the type of dataset that is being used. A dataset that has a high number of numeric columns representing static data like prices for an item for example can be imputed while a numeric column that stores the ID for an employer cannot. The advantage of imputing the data is that the clustering algorithm perform better when given more data to train the model instead of being given missing values. The first step when pre-processing the data is to validate the data stored in

each column and remove any redundant and inconsistent data. The spreadsheet software Microsoft Excel has built in components to validate the data. The Remove Duplicates component can be used on each column independently based on what the column type is for example if the column stores zip codes then it is possible for the data values to be same for certain rows. The Remove Duplicates component will remove all redundancies from the desired spreadsheet columns. The other component that can be used is Data Validation. The Data Validation component lets the user examine each column and specify the type of data that is expected to be present in the column, any other data types will be treated as an exception. The user can run this component on the spreadsheet to remove any data value from the column that does not belong to the expected data type. The Data Validation component allows the user to ignore blank or missing values if desired. The imputation of missing values can also be handled using specialized data mining tools lile Weka, IBM SPSS, R, RapidMiner etc.

The application allows the user to define the primary key of the table. The options for the primary key are designed to let the user either select a primary key from the existing columns of the spreadsheet or define a new primary key column. If the user decides to use a new primary key column then an auto increment column of type integer called SR_NO will be defined. On selecting the primary key the application will validate the data in the column to look for duplicate values. If there are duplicate values present then the column cannot be used as a primary key and the application will display an error to the user warning them about the same. Once all the details for importing the spreadsheet are entered, the framework will initialize the code to read in the details of the spreadhseet that include the column names and the data within. When reading in the column names the framework will also determine whether the columns are of type formula and store this information for use later. After the information is gathered the data is sent to the prediction algorithm to predict the data type, this operation is done per column. There are numerous data types supported by the H2 database. The broad classifications however are either string or numeric. The

string data types can be further classified into types like char, varchar, date, time, blob, boolean etc. while the numeric data types supported by H2 database can be classified into types like int, tinyint, smallint, bigint, decimal, double, real binary etc. Out of the various data types supported by the H2 database the framework currently identifies the following data types for string namely char, varchar, date, time and for the numeric data types the framework supports binary, int, tinyint, smallint, bigint, double and real. The framework broadly classifies each column entry into either string or numeric and then applies the prediction algorithm on the data. The prediction algorithm takes as input all the data entities from one column at a time. It then classifies each value as either a string or a number. Each type is processed differently. On being classified as a string the data entity is compared to regular expressions to check whether the value is a boolean, date or time. If that test fails the data is compared to the check whether it is either of type char or varchar. This can be determined based on whether the values are or same or different lengths. The same length strings are classified as char while the other ones are varchar. On the other hand if the data entity is classified as numeric then it is first compared through regular expressions to check if it is of binary form, if not it passes through a series of comparisons to check whether it is of type Integer, TinyInt, SmallInt or BigInt. The comparisons are done in a range based fashion. For example to check if the given data entity is of type Integer it is checked to see if it falls within the range of values for an Integer which are -2147483648 to 2147483647. Depending on the data type being checked this range will either increase or decrease.

The result is the predicted data type for each column. A column that has values that are made up of a formula are also treated the same way as other columns when predicting the data. After having the data type information for each column the data is imported into the H2 database table using those predicted data types and a create table SQL command is executed to create the table. The formula column information that was identified comes into play after this step. The column that holds the formula values is renamed in the database table to reflect the same. The column is renamed to $<$ Original column name $>$_$<$Arithmetic

operation >_< Spreadsheet column 1 >_< Spreadsheet column 2 >_ COLUMN _ FORMULA. This information comes in handy if the database table is exported back to an excel spreadsheet.

2.1.2 Exporting a Database Table

The reverse operation of importing the spreadsheet is exporting it back to an excel spreadsheet. In most real world scenarios a user would not commonly export a database table to an excel sheet unless the application demands it. The export operation simply allows the user to write the table back to an excel spreadsheet. This is a simple transfer of information from one form to another without requiring any form of conversion or data type predictions since the spreadsheet follows no defined structure the data semantics are not important. The only important aspect to consider when exporting is the formula columns. In order to not loose the relationships of the formula columns, the framework observes the column name for each column in the table and if it comes across a column with the string COLUMN_FORMULA in the end then it recognizes it as a formula column. It then pulls the information required to build the formula from the name before it creates the same column in the spreadsheet.

2.2 Functional and Transitive Dependencies

2.2.1 TANE Algorithm

The next step in the process is to identify the functional dependencies for the database table that is created. This is done using an implementation of the TANE algorithm. The algorithm will output the minimal functional dependencies for the created database table. the algorithm also determines the approximate dependencies for the table. An approximate dependency is one which does not hold in all tuples and may have errors or exceptions.

Some of the important terms and concepts used in the TANE algorithm are as follows,

Table 2.1: Example database table

ID	A	B	C
1	1	1	H
2	2	B	G
3	1	B	K

Consider the example database table from table 2.1, the table has three columns namely A, B, C and a forth column which acts as an ID column. The main concepts in the TANE algorithm include discovering the dependencies in terms of equivalence classes and partitions and then searching the space for functional dependencies.

- Partitions: Partitions are smaller subsets of the given relation which have similar attributes. From the given table the partitions for each column will be, $P(A) = \{\{1,3\},\{2\}\}$, $P(B) = \{\{1\},\{2,3\}\}$, $P(C) = \{\{1\},\{2\},\{3\}\}$.

- Equivalence Classes: The equivalence class here represents the data from the relation that is similar on a certain attribute. From the above table the equivalence class for column A is (1)=(3) since the first and third column share the same data which is also represented by using their respective ID's as $\{1,3\}$. Similarly for column B, its (2)=(3) or $\{2,3\}$.

- Search: The search algorithm in TANE uses sets of attributes starting with the smaller sets and moving on to larger ones. The algorithm stores these sets in a containment lattice. the containment lattice for the example database table will include $\{\phi, A, B, C, AB, AC, BC, ABC\}$. The algorithm will start with the single attributes and then move on to the larger ones. When the algorithm is looking at a particular set of attributes say Y, it test dependencies of the form, $Y \setminus \{Z\} \rightarrow Z$ where $Z \in Y$. Here $\{Z\}$ represents a unit set with only a single attribute while $Y \setminus \{Z\}$ means the set Y complement Z.

- Pruning attributes in the search space: In order to be classified as a minimal a dependency has to satisfy $Y \setminus \{Z\} \rightarrow Z$ where $Z \in Y$ and $Z \in C(Y \setminus \{B\})$ for all $B \in Y$. Applying this to a set $Y = \{D, E, F\}$ and assuming $\{F\} \rightarrow D$ is a valid dependency, we get $\{E, F\} \rightarrow D$ is not minimal.

- Stripped partitions: A striped partition is basically a partition of an attribute without the singleton equivalence classes. From the example table the stripped partition for attribute A is $\{\{1, 3\}\}$. Removing the singleton equivalence classes does not affect the dependency since the meaning stays the same.

- Error e: The measure e is used to determine whether the dependency is an approximate dependency. The error e is determined as the fraction of tuples to be removed from the relation R for the dependency $A \rightarrow B$ to hold in R.

The TANE algorithm works in a levelwise manner using the smaller levels or partitions first. The algorithm also considers results from the previous section to the next when computing the functional dependencies. The basic steps involved in the algorithm are,

- Start with level 1 where $L1 = \{\{A\}|A \in R\}$.

- while level is not equal to empty set

- Calculate functional dependencies

- Prune the partitions

- Generate the next level $L = L + 1$

2.3 Application Flow

The application is the front end to the user. The user has two options when first opening up the application. The import option can be used to select an Excel spreadsheet an import it to a database table while the export option is to be used for the reverse operation of

exporting a database table to an Excel spreadsheet. When importing the details needed to start the process are the database details and the path to the file to import. The next stpe is to choose the primary key which is either one of the existing columns of the database or a new auto increment column names SR_NO. If the user select an existing column to be used a primary key then that column is validated to check for presence of duplicate values. If found the application raises an error informing the user of the same. The next step is to use the gathered information and pass it to the framework in order to get the predicted data types for each column. Once this information has been computed by the framework the next step is to create a table using SQL commands in the H2 database. After the database is created the user can choose to calculate the functional and transitive dependencies for the table. If the option to do that is selected the framework uses the TANE algorithm to calculate the minimal functional dependencies based on the output of which the transitive dependencies are calculated. If the user chooses the option to export a database table to an excel spreadsheet then the framework will export the data for each column and also create formula columns if they exist in the table.

2.4 User Input

The framework requires that the user inputs database details like the username, password and the database name when importing or exporting. The only data that differs is when importing the user is required to input the path including the filename of the Excel sheet to be imported while when exporting the user has to input the path including the filename to export to along with the database table name to export. The paths for the files are expected to be locations that exist. Invalid path will result in failure. The same applies to the database connection, the user has to ensure that the database server is running before trying to establish a connection through the application. The only other scenario where user input is required is when the primary key is to be selected. The primary keys are displayed when the user clicks on the display columns option. The user can then choose to select an auto increment column or an existing column from the spreadsheet.

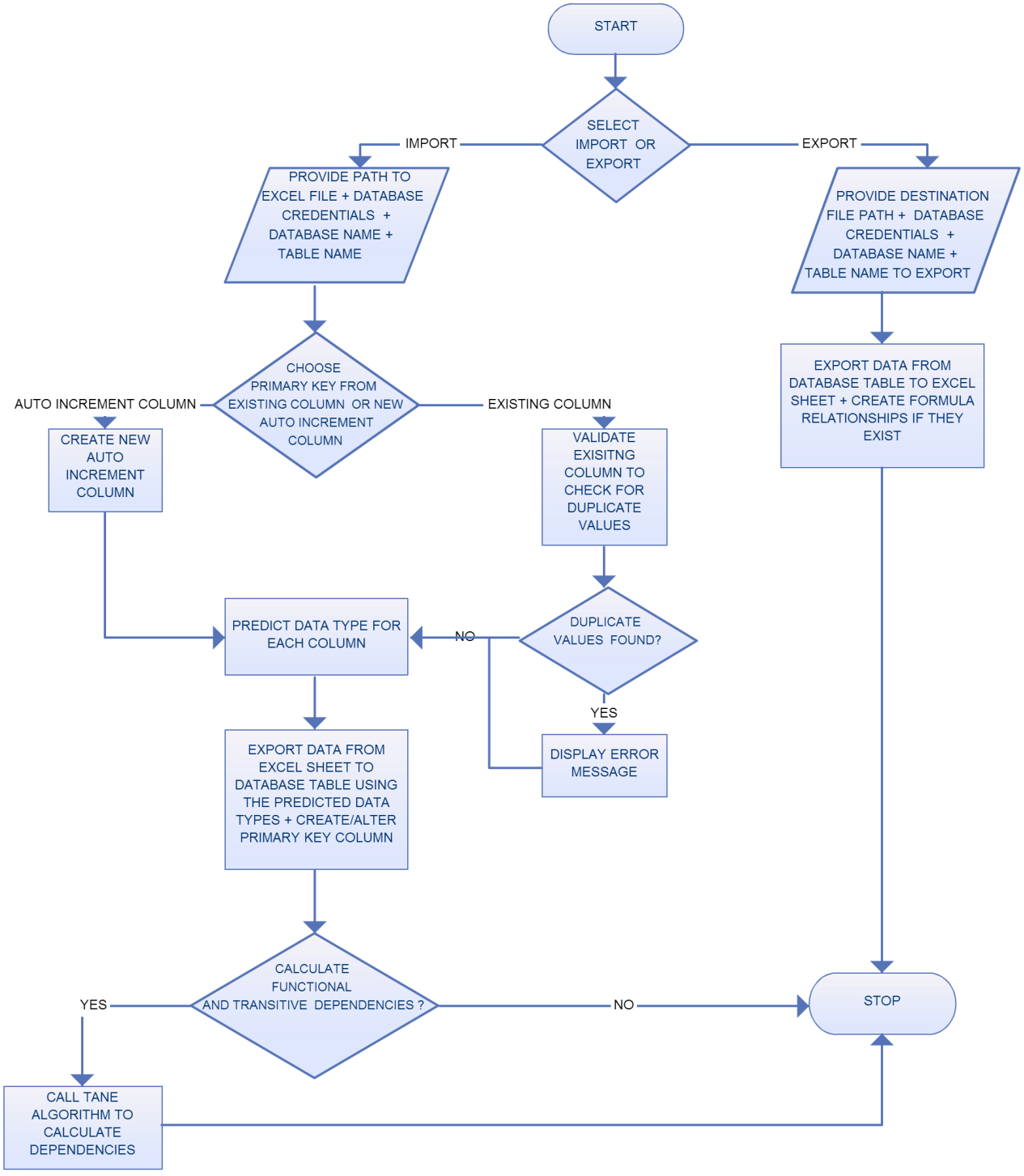

START
SELECT IMPORT OR EXPORT
IMPORT
EXPORT
PROVIDE PATH TO EXCEL FILE + DATABASE CREDENTIALS + DATABASE NAME + TABLE NAME
PROVIDE DESTINATION FILE PATH + DATABASE CREDENTIALS + DATABASE NAME + TABLE NAME TO EXPORT
CHOOSE PRIMARY KEY FROM EXISTING COLUMN OR NEW AUTO INCREMENT COLUMN
AUTO INCREMENT COLUMN
EXISTING COLUMN
EXPORT DATA FROM DATABASE TABLE TO EXCEL SHEET + CREATE FORMULA RELATIONSHIPS IF THEY EXIST
CREATE NEW AUTO INCREMENT COLUMN
VALIDATE EXISITNG COLUMN TO CHECK FOR DUPLICATE VALUES
PREDICT DATA TYPE FOR EACH COLUMN
NO
DUPLICATE VALUES FOUND?
YES
DISPLAY ERROR MESSAGE
EXPORT DATA FROM EXCEL SHEET TO DATABASE TABLE USING THE PREDICTED DATA TYPES + CREATE/ALTER PRIMARY KEY COLUMN
CALCULATE FUNCTIONAL AND TRANSITIVE DEPENDENCIES ?
YES
NO
STOP
CALL TANE ALGORITHM TO CALCULATE DEPENDENCIES

Figure 2.1: Process Flow for GUI application.

Chapter 3

Implementation

3.1 Languages and Tools

The framework is written in JAVA and implemented using Eclipse. The framework utilizes two externally referenced libraries for enabling some features. The first library is the JDBC driver for the H2 database, this is used to provide a connection from the framework to the database sever and to execute the various SQL commands required in order to create and/or manipulate the database table. The second library is the Apache POI API which is used to translate between JAVA and the Excel spreadsheet. The POI API provides a means to read and write Microsoft Office documents in this case the Excel spreadsheet.

3.2 Class Overview

3.2.1 Excel Component Overview

Figure 3.1 gives a high level view for the Excel component that is used for data type prediction. A detailed class diagram for the same can be found in Fig.1 of Appendix A. The main classes that comprise the Excel component of the framework are GUI, Excel and FindDataType. The GUI application built using JAVA Swing acts as the front end component to the user. The process connects to the Excel class, the excel class is responsible for reading and writing of the excel files. Excel class in turn connects to the FindDataType class which is responsible for predicting the data type for the columns in the Excel spreadsheet. The FindDataType class uses two classes as helpers to break down the input and enable easier

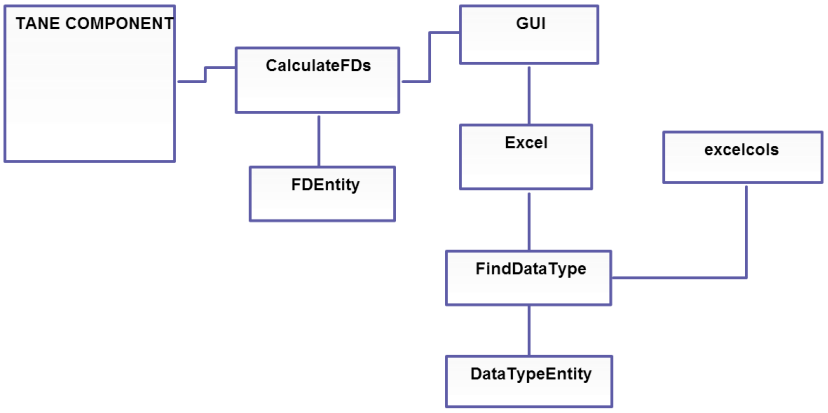


Figure 3.1: High level class diagram for the Excel Component.

storage of the data and also make it easier to understand the code. The first class, FindDataType initializes an instance of each data type before it starts the calculation. While executing the FindDataType class increases the count for each data type found and in the end calculates a percentage values for each data type before publishing the results. The second helper class is the excelCols class. This class is used by Excel to send in column information to the FindDataType class. The excelCols class stores information for each column, this information includes the data stored in the column, the column name, a boolean specifying whether the column is a formula in which case it also stores the formula, the two separate columns involved in the formula and the operation. Apart from this information the excelCols class also stores the predicted data type for each column. The second class is DataTypeEntity. This helper class is used to store data about each data type field. It stores information like the column name, the count for each data type for every column and the corresponding percentage values. Essentially when the framework receives a column to find the data type it initializes a DataEntityType for each possible data type for the column. After the column data has passed through the prediction algorithm, the data in each of the DataTypeEntity objects is examined to figure out the predicted data type for that particular column.

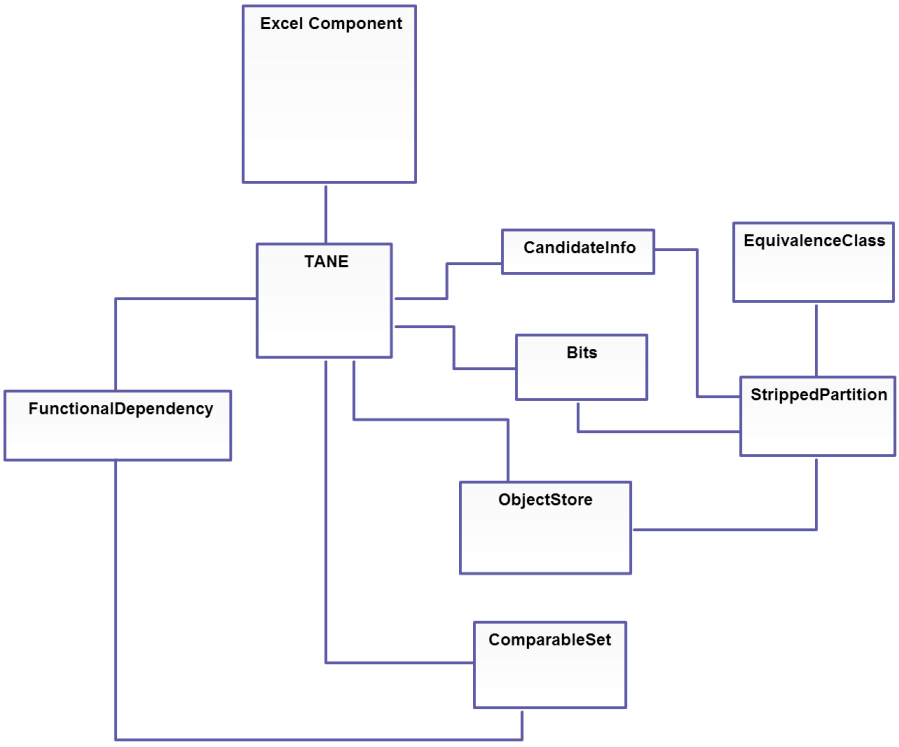


Figure 3.2: High level class diagram for the TANE Component.

3.2.2 TANE Component Overview

After the prediction process is complete the user can choose to compute the functional and transitive dependencies for the created database. When the user clicks the Calculate Functional Dependencies button on the GUI application, the application calls the CalculateFDs class. This class is responsible for calling into the TANE algorithm component and process the output. The output from the TANE algorithm comprises of the minimal functional dependencies for the table. The CalculateFDs class then examines the output and deconstructs it using the helper class FDEntity. The deconstruction process breaks the functional dependencies into their respective determinant and dependents namely the LHS and the RHS. This information is used to construct the list of transitive dependencies in the table. When the result is formed the CalculateFDs class passes the output back to the GUI to display it.

The next component of the framework is the TANE algorithm. Figure 3.2 gives a high level view for the TANE component that is used for functional dependency generation. A detailed class diagram for the same can be found in Fig.2 of Appendix A. The component is made up of smaller classes. The main class is the TANE class which is the heart of the algorithm. The TANE class also has other helper classes that are used to store data during execution. The TANE class follows the three main steps through the use of the methods in the class namely computeDependencies, prune and generateNextlevel. The computeDependencies method makes use of the helper classes StrippedPartition to store the partition information and EquivalenceClass to store the Equivalence class for each partition. The methods prune and generateNextLevel use the CandidateInfo class to store the RHS candidate set. To enable set operations the algorithm uses the ComparableSet class which has implementations for set operations like union, intersection, difference. These operations are needed during execution of the TANE algorithm. The algorithm makes use of the Bits and ObjectStore classes to stores the bit values for the RHS and LHS. When a functional dependency is found it is stored in the class FunctionalDependency.

3.3 User Input

The main screen for the framework is the application screen that has the input fields that require user input and the operations that they choose to select. Figure 3.3 shows the screen when the application is started.

If the user chooses to import a spreadsheet into a database table then they need to enter the required information like the database name with the credentials to connect to, along with the path for the file to import. Once that is done in order to select a primary key column they need to click on the Display Columns button which will display the columns in the spreadsheet along with the option to add an auto increment column. Figure 3.4 shows the screen after the user has entered the information on the GUI and selected the primary

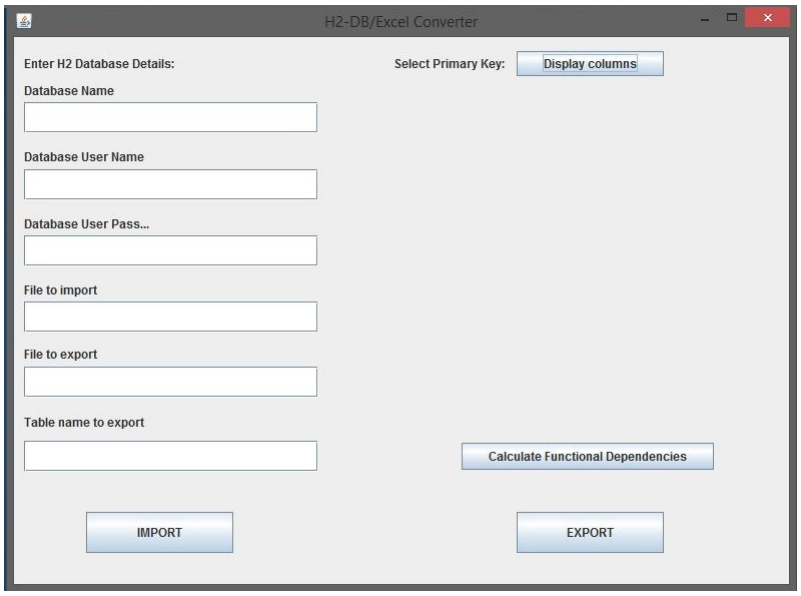


Figure 3.3: GUI application on first start.

key.

After the database has been created the user can choose to determine the functional and transitive dependencies for the table, this can be done by clicking on the Calculate Functional Dependencies button. Figure 3.5 shows an example of the output after the user has clicked the Calculate Functional Dependencies button.

Similarly when a user chooses to export a database table to an excel spreadsheet they need to enter the database name along with the credentials to connect to the database, the path export the excel file to and the database table name that is to be exported. Figure 3.6 shows the application screen after the user has entered the required information to export the table.

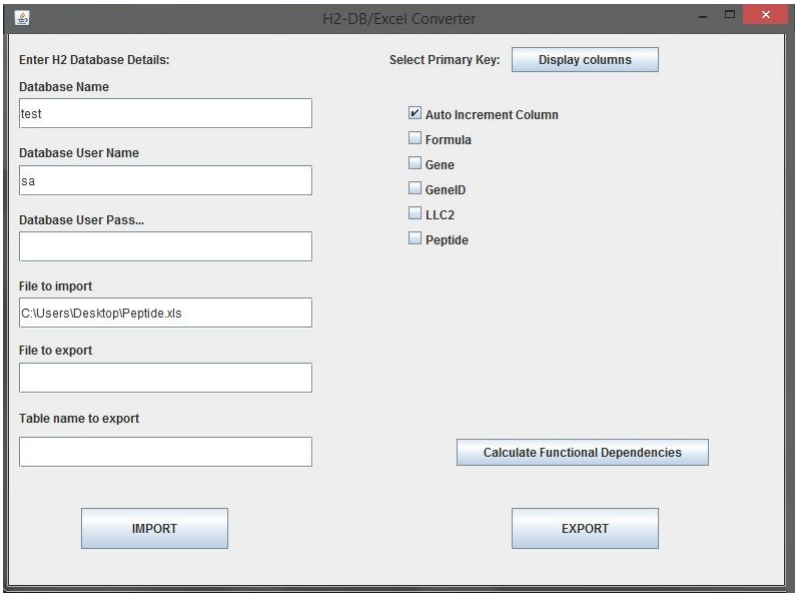


Figure 3.4: GUI application when importing a spreadsheet to the database.

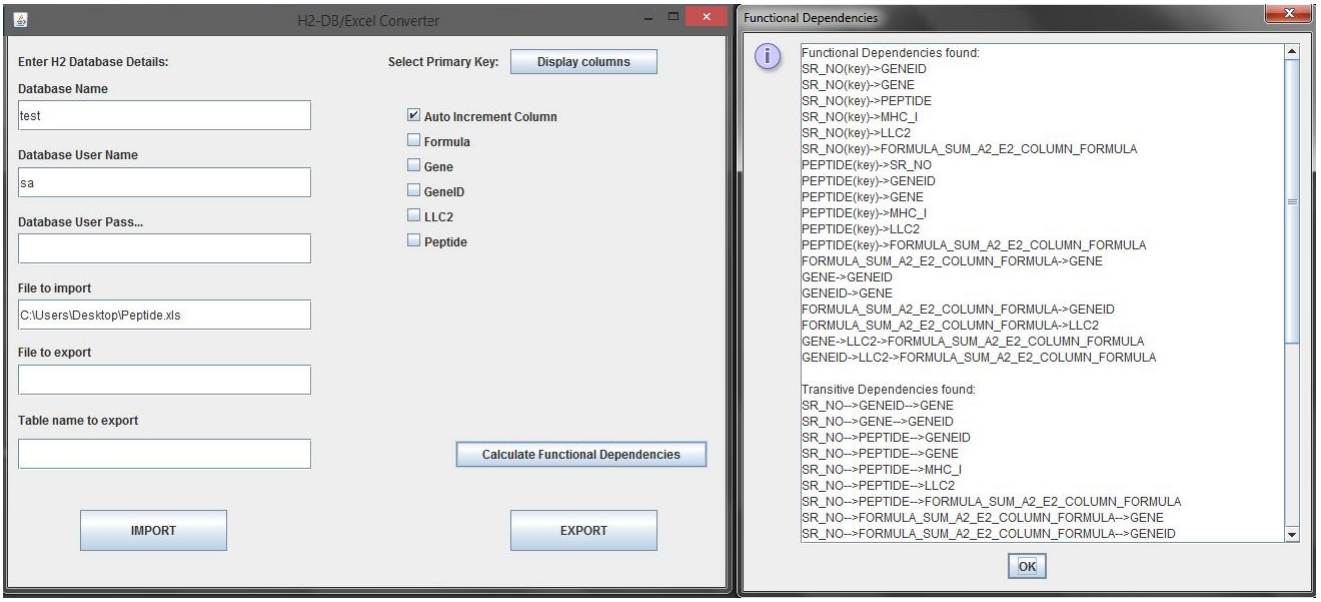


Figure 3.5: GUI application displaying output for Functional and Transitive dependencies.

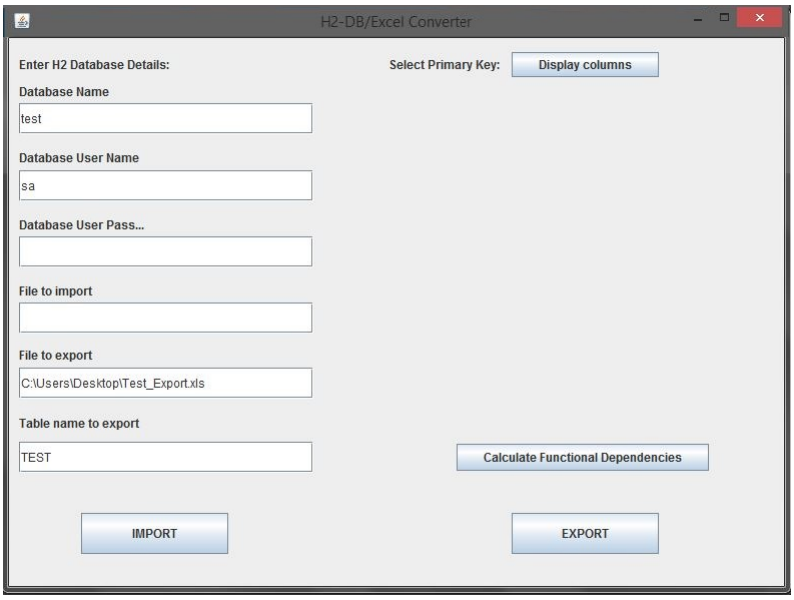


Figure 3.6: GUI application when exporting a database table to an excel spreadsheet.

Chapter 4

Analysis

4.1 Testing Environment

The application is written in JAVA. In order to test it, an executable jar file is created. The test environment is a Windows machine with Microsoft Office installed and an instance of the H2 database server running on it. The hardware specifications for the system are, Intel Core i7-2630QM CPU with 8 GB of RAM. The version of Microsoft Excel installed on the machine is 2013 however the application works with the older '.xls' format. The H2 database version used is 1.3.172.

4.2 Datasets

The application is tested by using various sized Excel datasets and running them through the framework. The sizes differ in terms of the number of columns and rows that the dataset contains. The datasets are applied in increasing order of rows. In order to determine the effectiveness of the framework the time taken to import the Excel spreadsheet into the database and vice versa is measured. The other measure to consider is the actual predicted data types for the columns in the spreadsheet while importing. The output of the TANE algorithm can be evaluated in a similar fashion by examining the functional dependencies found. The dependencies can be verified by the data semantics for the particular dataset. There were six datasets used in all for the analysis. A general description for the datasets follows,

- Dataset 1: The first dataset used was the Peptide dataset. The dataset has columns named GeneID, Gene, Peptide, MHC_I and LLC2. An additional formula column with the sum of GeneID and LLC2 was added later. The columns store numeric values in the GeneID and LLC2 columns, while the other two columns consist of alphanumeric values. The dataset contains 419 rows and 6 columns.

- Datset 2: The second dataset used is the Employee dataset. This dataset was created using random values. If contains fields that describe employees at a company, for e.g. Account, Name, Email, Company, Street Address, City, Zip, Phone and Birthdate. The dataset has 400 rows and 10 columns.

- Dataset 3: The third dataset is the Farmers dataset. It contains information about farmers from various states. It contains columns like FMID, MarketName, Website, Street, City, County, State, Zip. The dataset has 7179 rows and 8 columns.

- Dataset 4: The forth dataset is the Baby_Names_Beginning_2007 dataset. This dataset stores the different first names for babies born in the year range of 2007-2012. The columns in the dataset are Year, First_Name, County, Sex and Count. The dataset has 35217 rows and 5 columns.

- Dataset 5: The fifth dataset used is the Consumer_Complaints dataset. This dataset is a collection of the complaints issued by consumers about products related to the financial domain like Mortage, Debt Collection, Credit cards etc. The dataset has columns like Complaint_ID, Product, Sub_Product, Issue, Sub_Issue, State, Zip_Code, Submitted_via, Date_Received, Date_Sent_To_Company, Company, Company_Response and Timely_Response. The dataset had 65535 rows and 13 columns. The dataset holds the maximum number of rows allowed for an xls file.

- Dataset 6: Dataset 6 is a stripped down version of dataset 5 with only 50376 rows and 9 columns.

All the above described datasets use the .xls format. The datasets have been sourced from public dataset sharing sites like data.gov. In terms of cleaning and validation there are a few steps to execute before the dataset can be used. First and foremost all of the datasets have space separated names in the columns when they use more than one word for e.g. 'Complaint ID'. This is not supported by the H2 database, thus the column names for the datasets have to be edited to replace the space character with an underscore (Complaint_ID). The next step is to validate the data in the columns. For the most part the framework is tolerant of null or blank values in the columns but the problem arises when the table is being imported after the data types for each column have been predicted. The issue is that for columns that hold numeric values cannot be blank when writing to the H2 database. This is not permitted in relational databases. The columns with variations of the string values however are permitted to have null values. Hence as part of cleaning the datasets the columns have to be examined to determine if there are null values in columns with numeric data. To remedy the problem the entire row must be deleted. This is done to insure that the data semantics stay intact. If the null values are imputed with mean, mode or random values then the resulting dataset will loose meaning which will might in turn result in wrong predictions in the framework for data prediction as well as the TANE algorithm. Data validation also has to be performed on the columns to check for any irregular values. For e.g. a column like Complaint_ID that holds strictly numeric data cannot have characters or special characters in it. Similarly validation has to be performed on the columns that store values of type DATE in the spreadsheet. H2 database only recognizes dates when they are in the yyyy-MM-dd format. Hence any date columns in the datasets need to be explicitly converted to that format before being imported.

4.3 Import

The import operation imports the excel spreadsheet into a H2 database table. This operation was repeated on the 6 different datasets described above and the time taken for the process to complete was measured in seconds. The measured time indicates how good the algorithm

performs. This performance measure can be translated to the time complexity of the data type prediction algorithm. The time complexity of the algorithm signifies how well the algorithm performs under varying loads of input. Figure 4.1 shows the constructed graph with the time measurements for the import operations. In order to prove the validity of the translation from measured time to the time complexity the graph includes a trendline. The trendline is basically a regression line that is used to depict the relationship between two variables in this case the size of the datasets and the time taken to import them. The trendline is constructed in Excel using the graph tool. The trendline in figure 4.1 follows the time taken variable very closely. The trendline can be linear, exponential, polynomial, logarithmic, power, or moving average. The left vertical axis on the graph corresponds to the numbers of rows and columns while the right vertical axis signifies the time taken in seconds. The trenline has two attributes, the R^2 value and the equation representing the line. The R^2 value is of significance here. It is called the coefficient of determination. It signifies how well the trendline fits the given data plot. The R^2 value is a value between 0 and 1. The closer the value is to 1, the better the fit of the line. In the given graph the trendline has an R^2 value of 0.9992, in comparison the linear trendline has an R^2 value of 0.8686 while an exponential trendline displays a value of 0.9408. Thus it is observed that the R^2 value of 0.9992 which represents a polynomial function is the best fit trendline for the import graph.

4.4 Export

Figure 4.2 shows the graph for the export operation runtimes. The graph is similar to that of the import operation with the number of rows and columns on the left vertical axis and the time taken in seconds on the right vertical axis. The only difference on the time taken on seconds axis is the scale. Since the highest value for time taken is 4.727 the scale has a maximum value of 5 seconds. Like the import graph the export graph trendline with an R^2 value of 0.9897 follows a polynomial function.

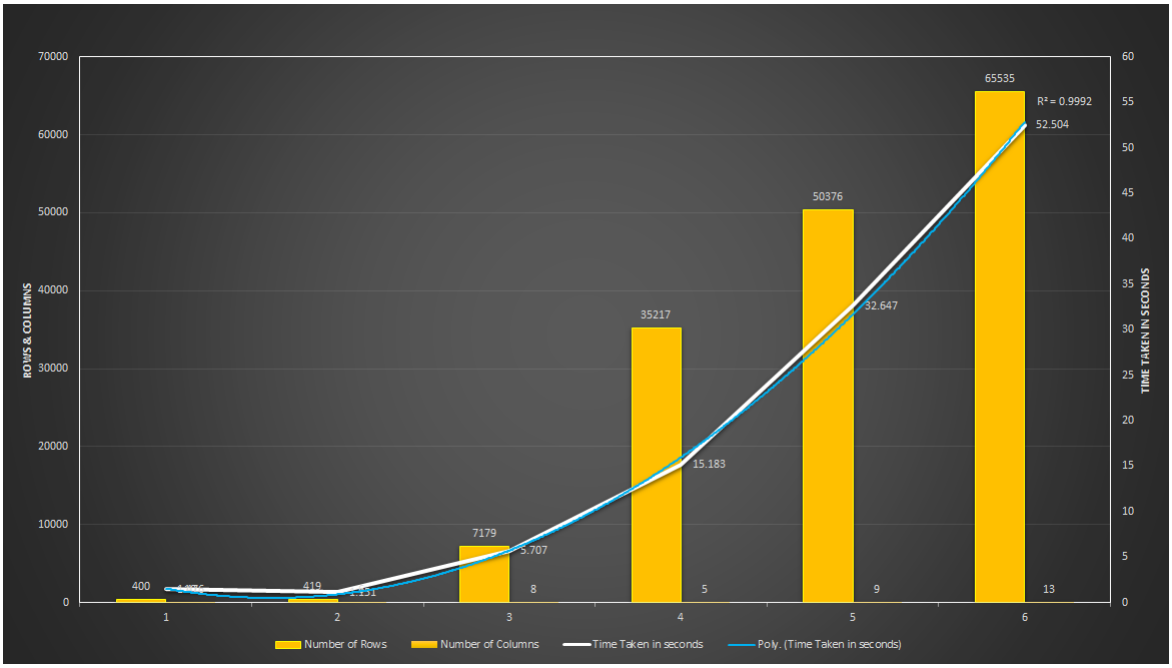


Figure 4.1: Time Measurements for the Import operation.

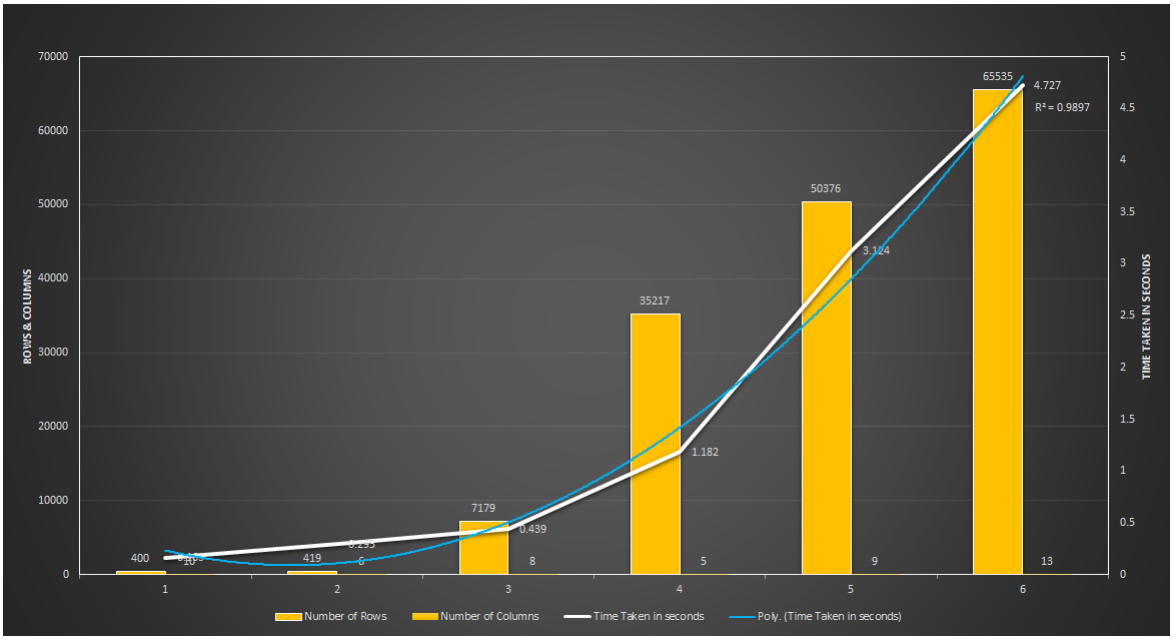


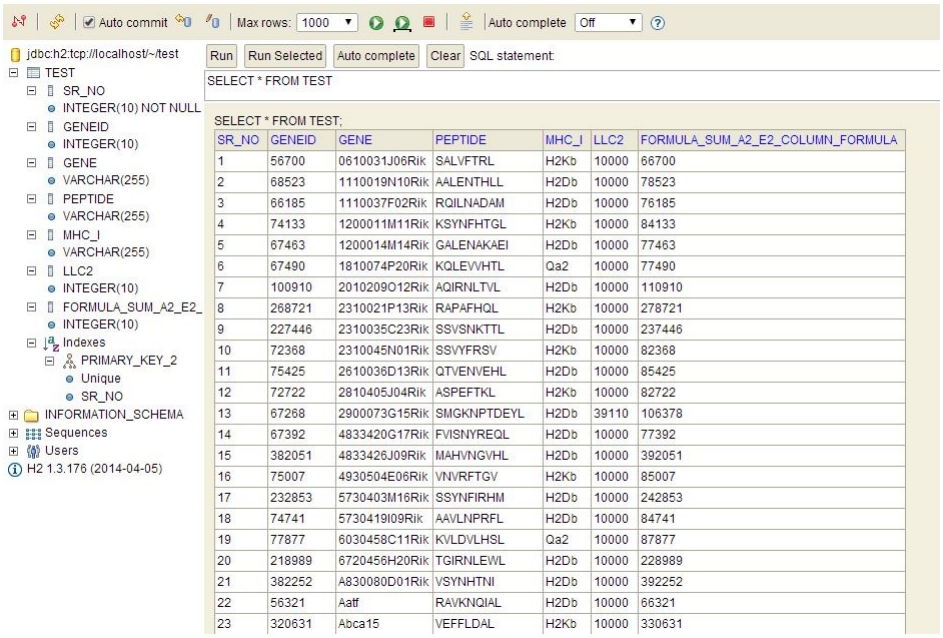
Figure 4.2: Time Measurements for the Export operation.

4.5 Hypothesis Evaluation

The results obtained during the experiments show that the approach taken by the framework has a polynomial time complexity. The framework does well to recognize the data types for the values within the columns of the spreadsheet as seen by running different datasets through it. Figure 4.3 shows an example of a Excel spreadsheet converted to a database table on the H2 database console. The results indicate that the framework accurately predicts the data type and thus defines a structure to the datasets by importing them into the database, also the primary key that was selected by the user is correctly set on the table. this can be verified using the data descriptions for the datasets used. For e.g. The dataset description for dataset 5 and 6 indicate that the Complaint_ID field is of type number and the date type for the Date_Received and Date_sent_to_company is of type Date while the rest of the columns are text. The data prediction algorithm recognizes the Complaint_ID column as an Integer, the State column as a Char, Date_received as a Date and the rest of the columns as a varchar. Similarly on the employee dataset that was created using random values of type Date for column Birthdate and all other columns are of type text, the algorithm predicts the data type for the Birthdate correctly as date while the other columns are predicted as having data type Varchar since the data in each column has variable length and contains alphanumeric values. The framework does not however support the entire range of data types supported by the H2 database, thus limiting the data types that are predicted.

The TANE component of the framework also works well in identifying the different functional dependencies between the columns of the database table. Figure 4.4 shows an example of the output from the TANE algorithm to predict the functional and transitive dependencies in the database table. The output also displays the transitive dependencies that are inferred from the out put of the TANE algorithm. traditionally to prove that the algorithm works the dataset would have to be passed through a data mining algorithm to define relationships on the columns of the dataset but for the purposes of the project based on the semantics of the data it can be derived that the functional dependencies do hold on

the database table. For e.g. For the Farmers dataset, the TANE algorithm outputs various functional dependencies. Most of them can be verified manually since the relationship is semantically correct. A functional dependency where all the columns depend on the FarmerID column independently is correct while a dependency where the county and zip and be used to determine the state or the Market name, Street address and the city can be combined to realize the zip.



SR_NO	GENEID	GENE	PEPTIDE	MHC_I	LLC2	FORMULA_SUM_A2_E2_COLUMN_FORMULA
1	56700	0610031J06Rik	SALVFTRL	H2Kb	10000	66700
2	68523	1110019N10Rik	AALENTHLL	H2Db	10000	78523
3	66185	1110037F02Rik	RQILNADAM	H2Db	10000	76185
4	74133	1200011M11Rik	KSYNFHTGL	H2Kb	10000	84133
5	67463	1200014M14Rik	GALENAKAEI	H2Db	10000	77463
6	67490	1810074P20Rik	KQLEVVHTL	Qa2	10000	77490
7	100910	2010209O12Rik	AQIRNLTVL	H2Db	10000	110910
8	268721	2310021P13Rik	RAPAFHQL	H2Kb	10000	278721
9	227446	2310035C23Rik	SSVSNKTTL	H2Db	10000	237446
10	72368	2310045N01Rik	SSVYFRSV	H2Kb	10000	82368
11	75425	2610036D13Rik	QTVENVEHL	H2Db	10000	85425
12	72722	2810405J04Rik	ASPEFTKL	H2Kb	10000	82722
13	67268	2900073G15Rik	SMGKNPTDEYL	H2Db	39110	106378
14	67392	4833420G17Rik	FVISNYREQL	H2Db	10000	77392
15	382051	4833426J09Rik	MAHVNGVHL	H2Db	10000	392051
16	75007	4930504E06Rik	VNVRFTGV	H2Kb	10000	85007
17	232853	5730403M16Rik	SSYNFIRHM	H2Db	10000	242853
18	74741	5730419I09Rik	AAVLNPRFL	H2Db	10000	84741
19	77877	6030458C11Rik	KVLDVLHSL	Qa2	10000	87877
20	218989	6720456H20Rik	TGIRNLEWL	H2Db	10000	228989
21	382252	A830080D01Rik	VSYNHTNI	H2Db	10000	392252
22	56321	Aatf	RAVKNQIAL	H2Db	10000	66321
23	320631	Abca15	VEFFLDAL	H2Kb	10000	330631

Figure 4.3: Database Table on the H2 Database Console.

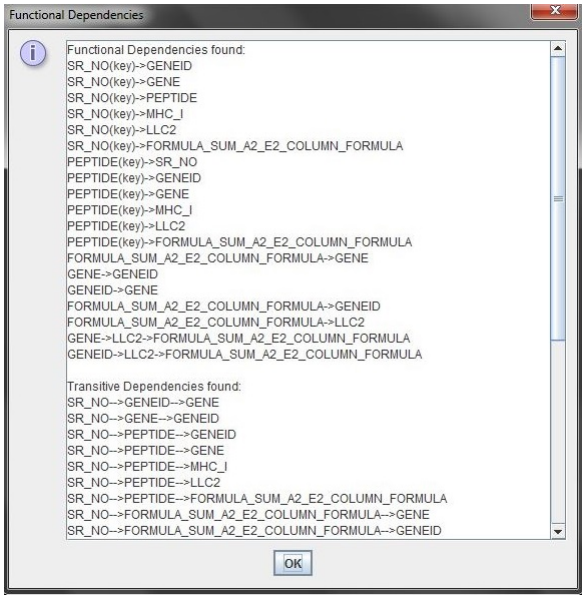


Figure 4.4: Functional and Transitive Dependencies for the Database Table.

Chapter 5

Conclusions

5.1 Current Status

The Excel spreadsheet to database converter has been implemented and tested to perform under varying loads of input through different excel spreadsheets. The framework consists of two parts, the data prediction algorithm and an implementation for a popular functional dependency prediction algorithm called TANE. The data prediction algorithm is verified to perform in polynomial time complexity. which is better than an exponential time given that the algorithm looks at each column linearly followed by each row for that particular column.

5.2 Future Work

There are several areas that the framework can be improved in.

5.2.1 Support For Newer Versions of Excel

Currently the framework only supports the .xls extension which is the 1997-2003 spreadsheet format for excel. The framework can be expanded to include more versions of Excel like the newer .xlsx format. The framework can also be made more flexible by supporting spreadsheet software packages apart from Microsoft Excel.

5.2.2 Faster Data Prediction Algorithm

The data prediction algorithm currently displays polynomial time complexity. The algorithm can be tweaked to perform much faster and reduce the time complexity to be linear or even logarithmic. This can be done by making the algorithm perform the data prediction in parallel. MapReduce is a framework that allows a program to be broken down into parallel by assigning the same task to multiple threads or nodes and executing them in parallel in what is called the mapping step. When the execution cycles are finished the final result is combined in the reduce step. This will drastically reduce the amount of time taken by the algorithm to process a dataset.

5.2.3 Support For More Databases and Data Types

The framework can be exapanded to work with other relational databases apart from the H2 database. It can also be expanded to include support for more data types that have not been implemented as part of this project.

5.2.4 Additional User Experience Changes

The user application or GUI can be expanded to include more user interaction for e.g. the table name for an import operation is currently limited to the name TEST, this can be changed by accepting the desired table name as input from the user. The application can also be made more fault tolerant in the sense that it can warn the user about failure scenarios by displaying better errors on the GUI.

5.3 Conclusions

The framework provides the user with a simple user interface to convert an Excel spreadsheet to a database spreadsheet by defining a structure to the data stored within. Once converted to a relational database schema the spreadsheet data will inherit all the properties of a database table including features like security, multi user access along with the fact that

the amount of data being stored will increase drastically. With the added capability of recognizing the functional and transitive dependencies in the database table, the user has the option to normalize the table thus making it less prone to the different anomalies. With the lack of other solutions that provide the same feature the framework is an attractive option for users who need to switch from working with a spreadsheet ans step into the database domain.

Bibliography

[1] Jácome Cunha, João Saraiva, and Joost Visser. From spreadsheets to relational databases and back. In *Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation*, pages 179–188. ACM, 2009.

[2] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, 42(2):100–111, 1999.

[3] Vu Hung, Boualem Benatallah, and Regis Saint-Paul. Spreadsheet-based complex data transformation. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1749–1754. ACM, 2011.

[4] Jerzy Tyszkiewicz. Spreadsheet as a relational database engine. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 195–206. ACM, 2010.

[5] Andrew Witkowski, Srikanth Bellamkonda, Tolga Bozkaya, Gregory Dorman, Nathan Folkert, Abhinav Gupta, Lei Shen, and Sankar Subramanian. Spreadsheets in rdbms for olap. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 52–63. ACM, 2003.

Appendix A

UML Diagrams

Figure A.1 shows the class diagram for the Excel component of the framework.

Figure A.2 shows the class diagram for the TANE component of the framework.

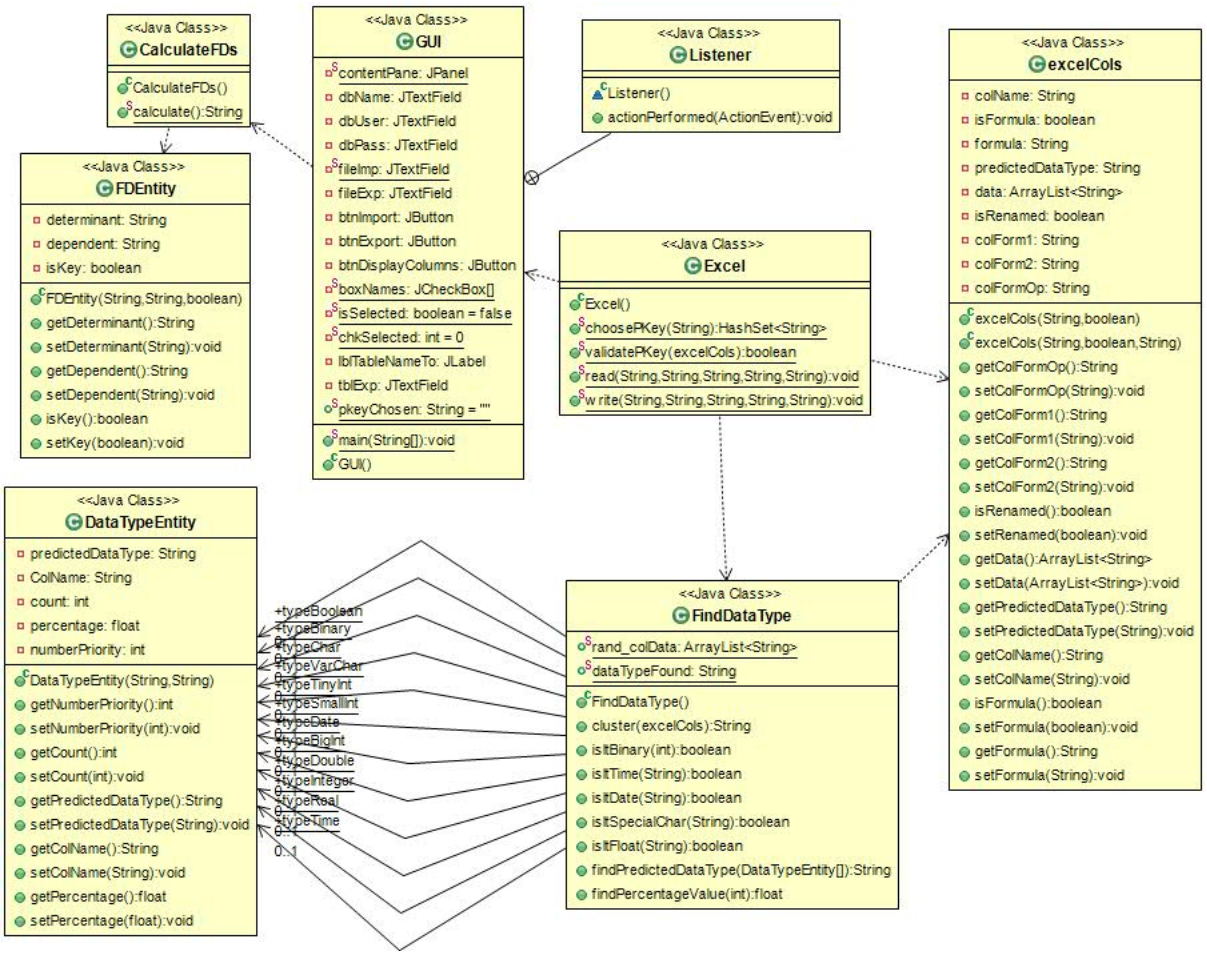


Figure A.1: Excel Component Class Diagram.

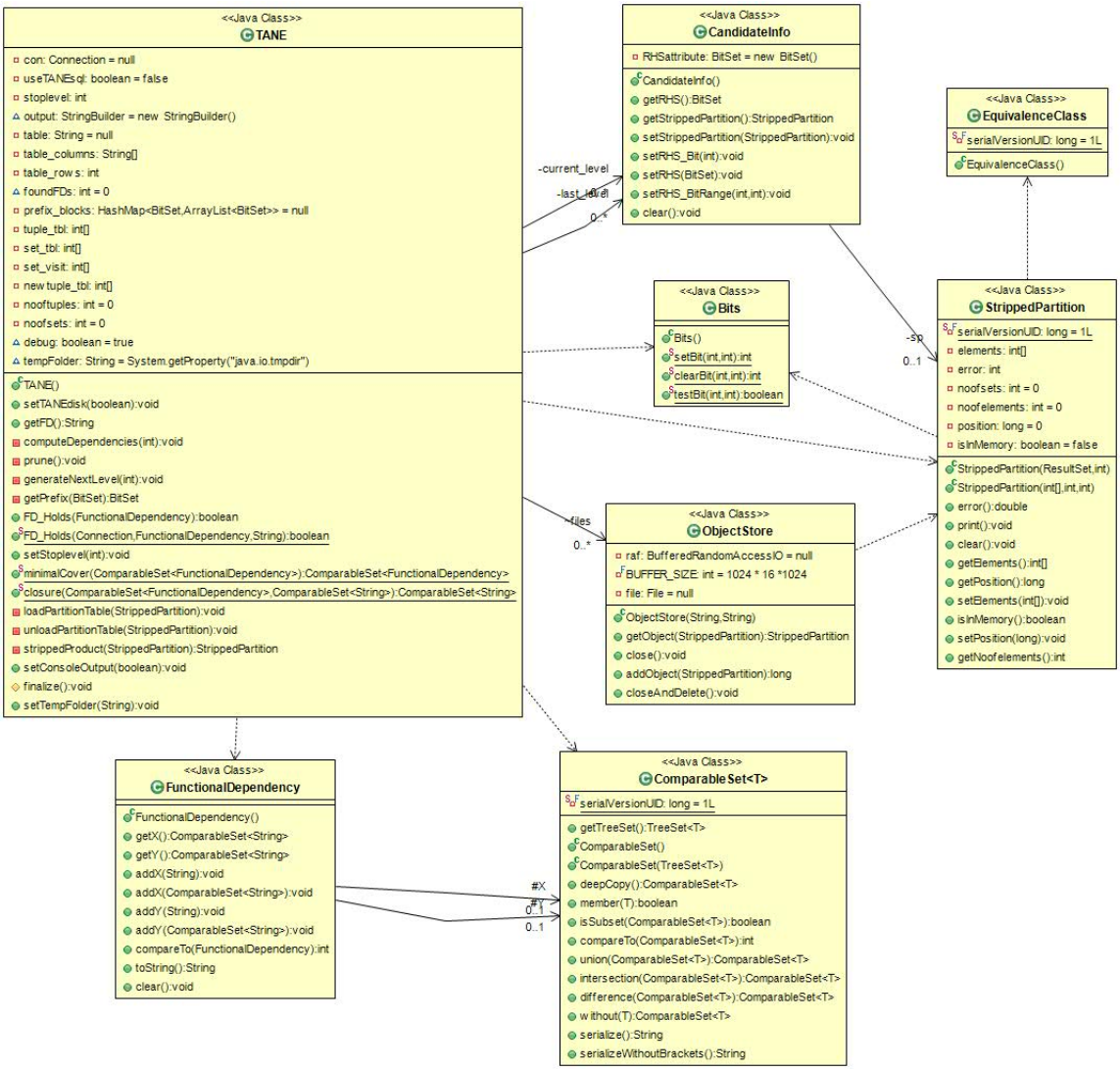


Figure A.2: TANE Component Class Diagram.

Appendix B

Code Listing

The code listing for the project is available on the attached disc.

Appendix C

User Manual

C.1 Installation

The GUI application for the framework is designed in JAVA. It is exported as an executable jar file. The jar file is platform independent, meaning it can be run on any operating system like Windows or Linux provided they have a JVM environment setup. Once the application jar is copied onto the machine, it can be simply run by double clicking on the jar and executing it like any other application. The only requirement here is that the irrespective of the operating system being used there should be a valid instance of Microsoft Excel and the H2 database server running on the machine. Once opened the application is ready to be used to for import or export operations.

C.2 User Interaction

The main screen for the framework is the application screen that has the input fields that require user input and the operations that they choose to select. Figure C.1 shows the screen when the application is started.

If the user chooses to import a spreadsheet into a database table then they need to enter the required information like the database name with the credentials to connect to, along with the path for the file to import. Once that is done in order to select a primary key column they need to click on the Display Columns button which will display the columns in the spreadsheet along with the option to add an auto increment column. Figure C.2 shows

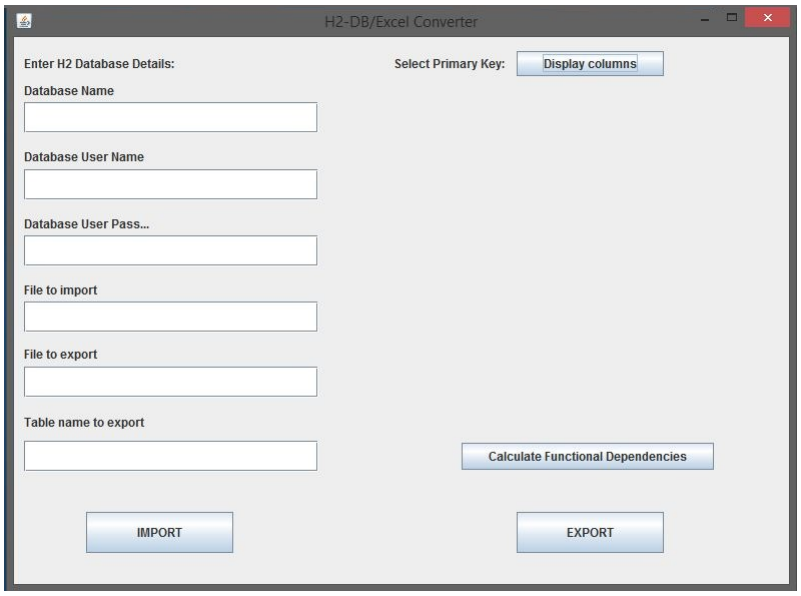


Figure C.1: GUI application on first start.

the screen after the user has entered the information on the GUI and selected the primary key.

After the database has been created the user can choose to determine the functional and transitive dependencies for the table, this can be done by clicking on the Calculate Functional Dependencies button. Figure C.3 shows an example of the output after the user has clicked the Calculate Functional Dependencies button.

Similarly when a user chooses to export a database table to an excel spreadsheet they need to enter the database name along with the credentials to connect to the database, the path export the excel file to and the database table name that is to be exported. Figure C.4 shows the application screen after the user has entered the required information to export the table.

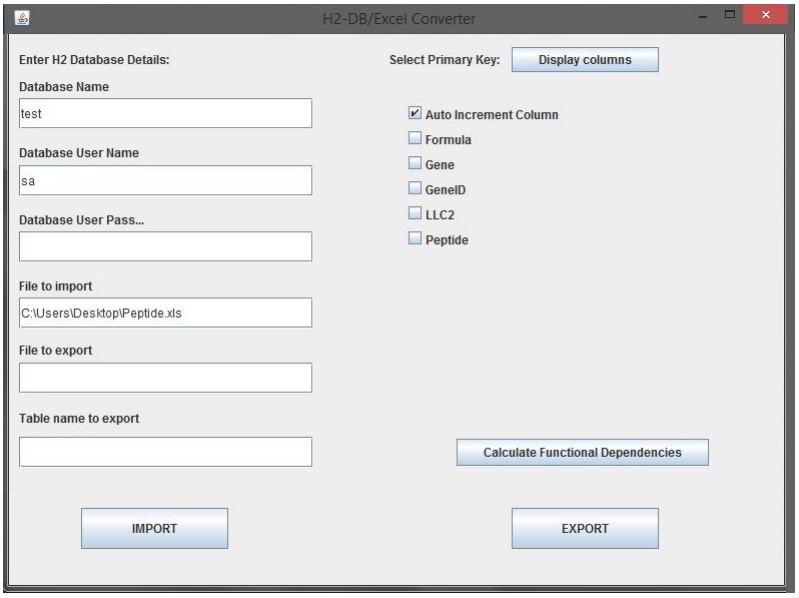


Figure C.2: GUI application when importing a spreadsheet to the database.

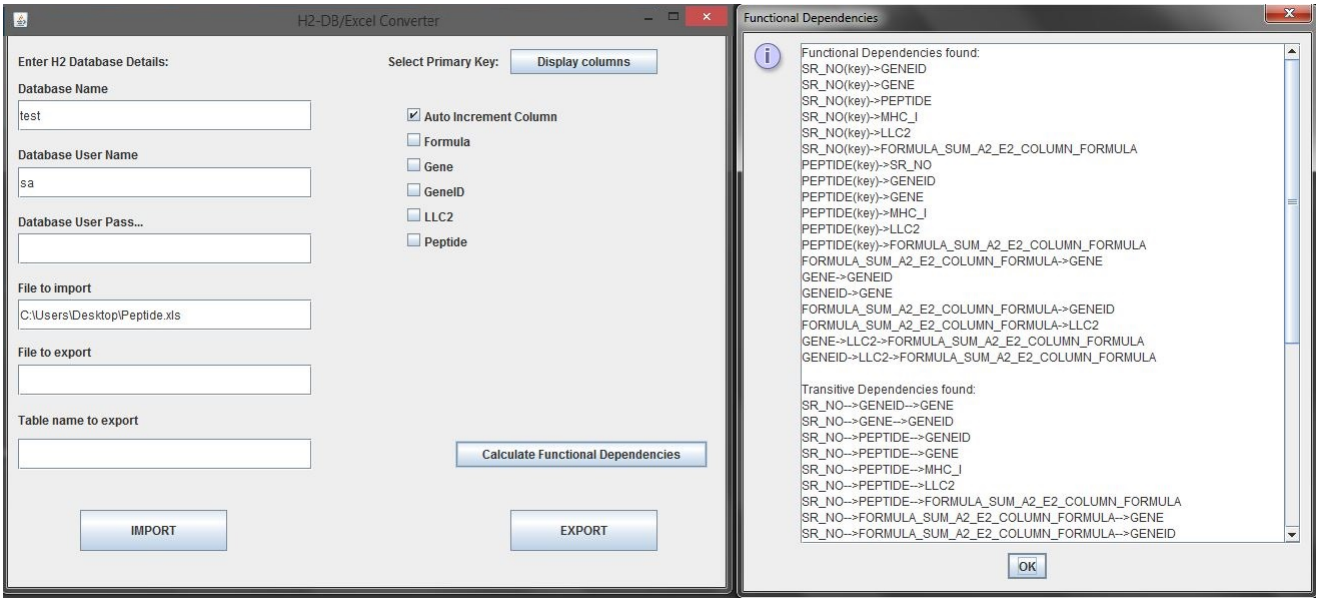


Figure C.3: GUI application displaying output for Functional and Transitive dependencies.

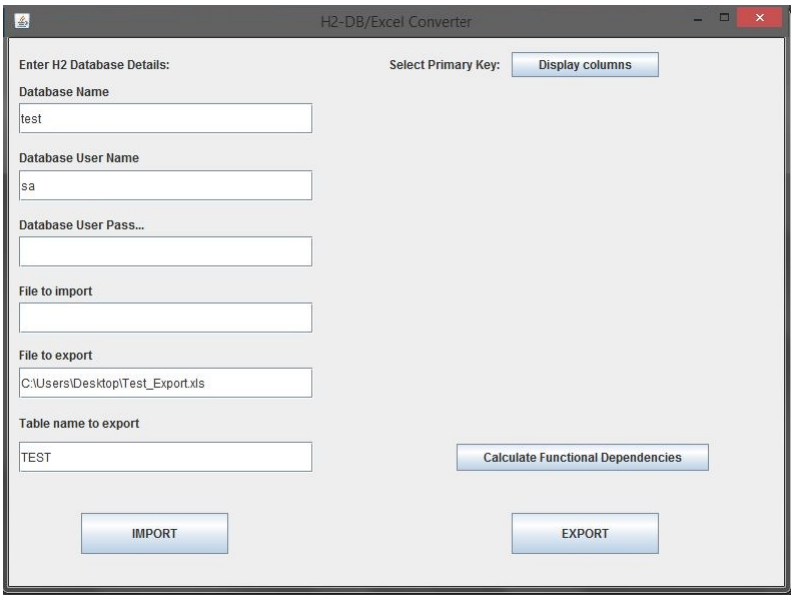


Figure C.4: GUI application when exporting a database table to an excel spreadsheet.

C.3 Error Handling

Currently the application is designed to handle two types or errors and display appropriate messages for the same. The first error is when a user is trying to either an import or export operation and has not entered all the required information for that operation then the application will throw an error notifying the user that there is missing information. The second set of errors is related to the primary key selection. When a user selects multiple primary keys the application will warn the user that it is an invalid operation also if the selected column for the primary key does not hold unique values then the application will display a warning to the user notifying them about the same. Figure C.5 shows the import error. Figure C.6 shows the export error. Figure C.7 shows the multiple primary key selection error while figure C.8 shows the invalid primary key column error.

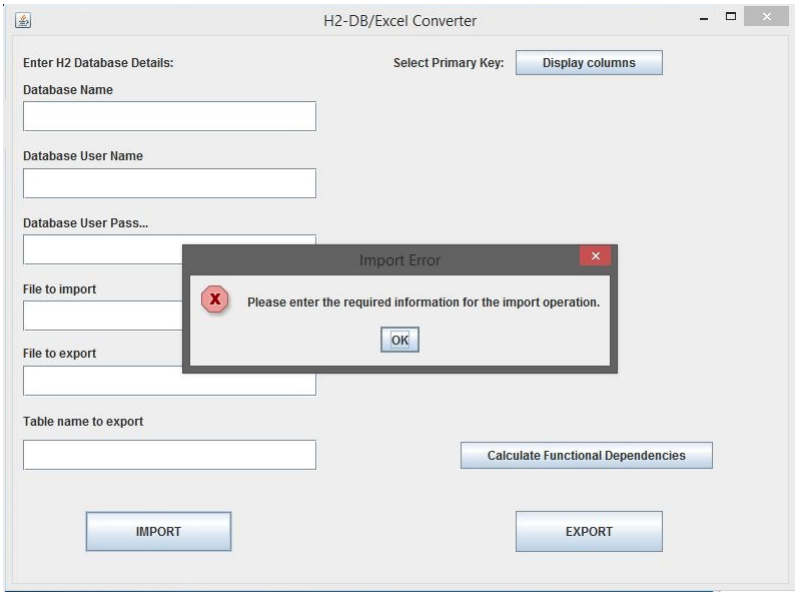


Figure C.5: Import Error.

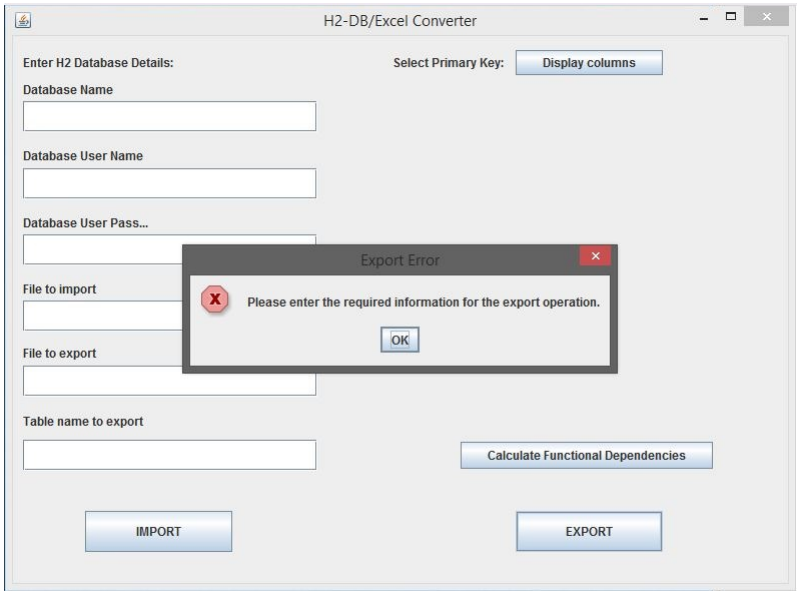


Figure C.6: Export Error.

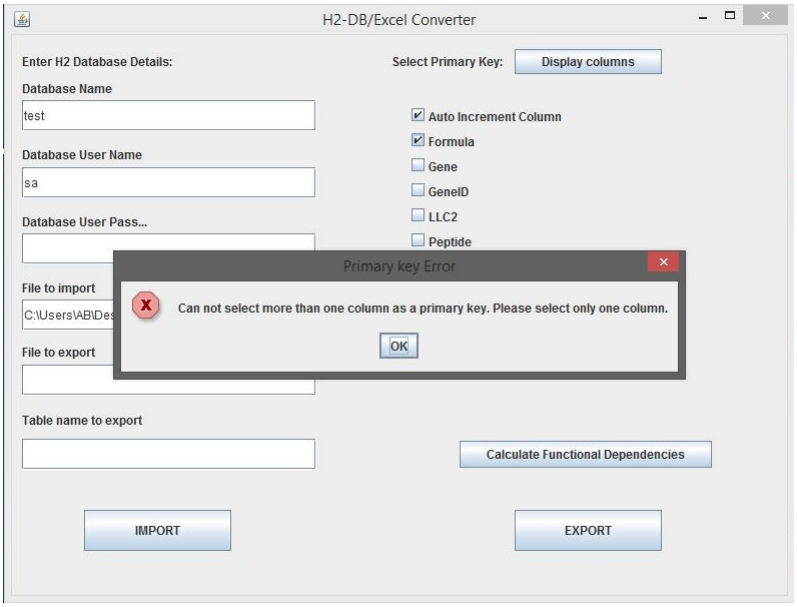


Figure C.7: Multiple Primary Keys Chosen Error.

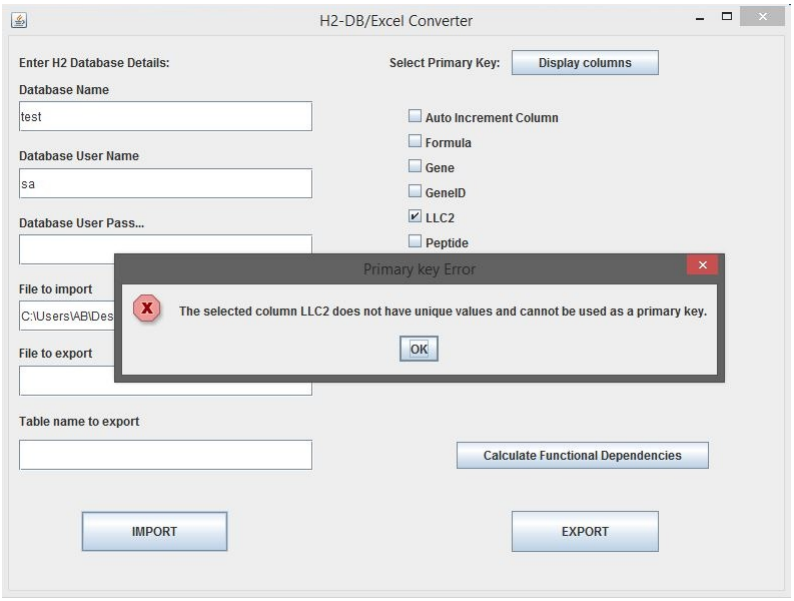


Figure C.8: Invalid Primary Key Error.